Examen PP

31.05.2019

ATENȚIE: Aveți 2 ore · 1-9: 10p; 10: 30p · 100p pentru nota maximă · **Justificați** răspunsurile!

---

1. Reduceți expresia lambda $E = (y\ \ (\lambda x.\lambda x.x\ \ (\lambda y.y\ \ y)))$

2. Se dă următorul cod Racket:
   ```
   (define computation (λ () (equal? 5 5)))
   (define (f x) (and (> x 5) (computation)))
   (filter f '(1 3 5 7 9))
   ```
   (a) De câte ori se apelează funcția `equal?` ?
   (b) Rescrieți codul pentru `computation` și pentru `f` folosind promisiuni (pentru întârzierea lui `computation`) și răspundeți din nou la întrebarea (a).

3. Dată fiind o listă de liste de numere `LL`, scrieți în Racket codul care produce sublista lui `LL` în care pentru toate elementele `L` suma elementelor este cel puțin egală cu produsul lor. E.g. pentru `L = ((1 2 3) (1 2) (4 5) (.5 .5))` rezultatul este `((1 2 3) (1 2) (0.5 0.5))`. Nu folosiți recursivitate explicită.

4. Sintetizați tipul următoarei funcții în Haskell: `f = map (++)`

5. (a) Câte aplicații ale funcției de incrementare sunt calculate pentru evaluarea expresiei Racket `(length (map add1 '(1 2 3 4 5 6 7 8 9 10)))` ?
   (b) Dar pentru expresia Haskell `length $ map (+ 1) [1 .. 10]` ?

6. Supraîncărcați în Haskell operatorii `(+)` și `(*)` pentru valori booleene, pentru a surprinde operațiile de *sau*, respectiv *și* logic.

7. Transformați propoziția „Nu mor caii când vor câinii.” în logică cu predicate de ordinul întâi, folosind predicatele `caii_mor(când)` și `câinii_vor(când)`.

8. Se dă programul Prolog:
   `p(_, [], []).`
   `p(A, [A|B], B) :- !.`
   `p(A, [B|C], [B|D]) :- p(A, C, D).` Ce relație există între cele 3 valori `X`, `Y`, `Z`, dacă `p(X, Y, Z)` este adevărat?

9. Se dau următoarele relații genealogice prin predicatul `c(Parinte, Copil)`. Implementați predicatul `veri(X, V)`, care leagă `V` la lista de veri ai lui `X` (dacă există). De exemplu, pentru definițiile de mai jos, interogarea `veri(faramir, V)` leagă `V` la `[jenny, karl, ninel, octav]`.
   ```
   c(alex, celia).  c(alex, delia).  c(alex, marcel).
   c(barbra, celia).  c(barbra, delia).  c(barbra, marcel).
   c(delia, faramir).
   c(ion, jenny).  c(ion, karl).  c(celia, jenny).  c(celia, karl).
   c(marcel, ninel).  c(marcel, octav).
   ```

10. PROBLEMA (Poate fi implementată în orice limbaj studiat la PP.) Se urmărește implementarea unui *hash set*, care reprezintă o mulțime grupând valorile în bucket-uri, fiecare bucket fiind unic determinat de hash-ul valorilor din bucket (toate valorile din bucket au același hash). Hashul unei valori va fi dat de funcția `hash`, respectiv predicatul `hash(+V, -Hash)`.
    (a) Descrieți reprezentarea *hash set*-ului. Pentru Haskell, dați definiția tipului de date polimorfic. Definiți funcția/predicatul `values'`, care extrage lista tuturor valorilor asociate cu un *hash*.
    (b) Definiți funcția/predicatul `insert'`, pentru adăugarea unei valori.
    (c) Definiți funcția/predicatul `map'`, care aplică o funcție/predicat pe fiecare valoare din *hash-set*. NOTĂ: în Prolog, `map'` va aplica întotdeauna un același predicat `p(+VIn, -VOut)`.